Data Converter C++ API

CipherLab Co., Ltd.

Version 2.3

**Revison Note**

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Revised By** | **Revised Note** |
| 2.1 | 2015/1/16 | Susan | Initial Draft |
| 2.2 | 2015/10/22 | Susan | 1. Update AGX Type Enumeration and sample code to support Forge Ag v2.0 and above. 2. Add ProgressCallback Class reference |
| 2.3 | 2016/07/18 | Susan | Add “How to create a C++ Console Project in Linux” section |
| 2.3 | 2016/08/11 | Susan | Add compatibility of GNU C++ Compiler version |

內容

[Introduction 7](#_Toc458693077)

[How to create a C++ console project in Windows 8](#_Toc458693078)

[How to create a C++ Console Project in Linux 11](#_Toc458693079)

[For amd64: 11](#_Toc458693080)

[For i386 16](#_Toc458693081)

[Namespace 23](#_Toc458693082)

[CipherLab::DataConverter 23](#_Toc458693083)

[Enumerations 24](#_Toc458693084)

[enum AGXType 24](#_Toc458693085)

[enum BasicProgDbfFileName 24](#_Toc458693086)

[enum ConversionStatus 24](#_Toc458693087)

[enum FileType 24](#_Toc458693088)

[enum ForgeAgProgDbfFileName 25](#_Toc458693089)

[enum ProgrammingTool 25](#_Toc458693090)

[enum WrongFormatAction 25](#_Toc458693091)

[Class Documentation 26](#_Toc458693092)

[AGX Class Reference 26](#_Toc458693093)

[static int DetermineFormatType (const char \*path) 26](#_Toc458693094)

[static ForgeAgProgDataFormat \* Read (const char \* path, int &counter) 27](#_Toc458693095)

[ApiVersion Class Reference 28](#_Toc458693096)

[int GetApi\_major\_version ( ) 28](#_Toc458693097)

[int GetApi\_minor\_version ( ) 28](#_Toc458693098)

[int GetApi\_build\_version ( ) 28](#_Toc458693099)

[const char \* GetApiVersion ( ) 28](#_Toc458693100)

[BasicProgDataFormat Class Reference 28](#_Toc458693101)

[BasicProgDataFormat () 29](#_Toc458693102)

[~BasicProgDataFormat () 29](#_Toc458693103)

[BasicProgDataFormat ( const char \* path, BasicProgDbfFileName dbfFileName ) 29](#_Toc458693104)

[BasicProgDataFormat ( const char \* path, BasicProgDbfFileName dbfFileName, char delimiter ) 29](#_Toc458693105)

[ConversionResult Class Reference 31](#_Toc458693106)

[int getNumberOfErrorRecords () 31](#_Toc458693107)

[const char \* getOutputFilePaths (int index) 31](#_Toc458693108)

[int getNumberOfOutputFiles () 31](#_Toc458693109)

[ConversionResult () 31](#_Toc458693110)

[~ ConversionResult () 32](#_Toc458693111)

[CProgDataFormat Class Reference 33](#_Toc458693112)

[CProgDataFormat () 33](#_Toc458693113)

[~CProgDataFormat () 33](#_Toc458693114)

[CProgDataFormat ( const char \*path, const char \* dbfFileName ) 33](#_Toc458693115)

[CProgDataFormat ( const char \* path, const char \* dbfFileName, char delimiter ) 34](#_Toc458693116)

[DataFormat Class Reference 35](#_Toc458693117)

[DataFormat (int maxFields, char delimiter=0x00) 35](#_Toc458693118)

[~DataFormat () 35](#_Toc458693119)

[const char \*getPath () 35](#_Toc458693120)

[void setPath (const char \*path) 35](#_Toc458693121)

[char getDelimiter () 36](#_Toc458693122)

[void setDelimiter (char ch) 36](#_Toc458693123)

[bool isRecordDelimited () 36](#_Toc458693124)

[const char \* getDbfFileName () 36](#_Toc458693125)

[void setDbfFileName (const char \* fileName) 36](#_Toc458693126)

[int getNumberOfRecordFields () 36](#_Toc458693127)

[int getMaxRecordFields () 37](#_Toc458693128)

[int getKeyCount () 37](#_Toc458693129)

[RecordField GetRecordFieldAt (int index) 37](#_Toc458693130)

[void EditRecordFieldPosition (int index, int position) 37](#_Toc458693131)

[void EditRecordFieldLength (int index, int length) 37](#_Toc458693132)

[void EditRecordFieldIsKey (int index, bool isKey) 37](#_Toc458693133)

[void ClearRecordFields () 38](#_Toc458693134)

[void AddRecordField (RecordField recordField) 38](#_Toc458693135)

[void ParseRecordFields (const char \* inputFilePath) 38](#_Toc458693136)

[Dbf2TxtConverter Class Reference 39](#_Toc458693137)

[Dbf2TxtConverter (ProgrammingTool tool, const char \*source, const char \* target) 39](#_Toc458693138)

[void Convert (ProgressCallBack \*cb) 39](#_Toc458693139)

[void ConvertAsync ( ProgressCallBack \* cb ) 39](#_Toc458693140)

[void CancelAsync () 39](#_Toc458693141)

[ErrorRecord Class Reference 41](#_Toc458693142)

[ErrorRecord ( string fileName, long recordNum, string recordText, bool isSkip ) 41](#_Toc458693143)

[ErrorRecord () 41](#_Toc458693144)

[~ErrorRecord () 41](#_Toc458693145)

[const char \* getFileName () 41](#_Toc458693146)

[long getRecordNum () 41](#_Toc458693147)

[const char \* getRecordText () 42](#_Toc458693148)

[bool isSkipped () 42](#_Toc458693149)

[ForgeAgProgDataFormat Class Reference 43](#_Toc458693150)

[ForgeAgProgDataFormat () 43](#_Toc458693151)

[~ForgeAgProgDataFormat () 43](#_Toc458693152)

[ForgeAgProgDataFormat ( const char \*path, ForgeAgProgDbfFileName dbfFileName ) 43](#_Toc458693153)

[ForgeAgProgDataFormat ( const char \* path, ForgeAgProgDbfFileName dbfFileName, char delimiter ) 44](#_Toc458693154)

[formatException Class Reference 45](#_Toc458693155)

[formatException (void) 45](#_Toc458693156)

[formatException (string message, string info="") 45](#_Toc458693157)

[~formatException (void) throw () 45](#_Toc458693158)

[string getMessage () 45](#_Toc458693159)

[RecordField Class Reference 46](#_Toc458693160)

[RecordField (int position, int length, bool isKey) 46](#_Toc458693161)

[RecordField (int position, int length) 46](#_Toc458693162)

[RecordField () 46](#_Toc458693163)

[ProgressCallBack Class Reference 47](#_Toc458693164)

[ProgressCallBack(void) 47](#_Toc458693165)

[void SetCallBack(CBFun fun) 47](#_Toc458693166)

[Txt2DbfConverter Class Reference 48](#_Toc458693167)

[Txt2DbfConverter ( BasicProgDataFormat & dataFormat, const char \* targetDir, WrongFormatAction actionTakenWhenWrong ) 48](#_Toc458693168)

[Txt2DbfConverter ( CProgDataFormat & dataFormat, const char \* targetDir, WrongFormatAction actionTakenWhenWrong ) 48](#_Toc458693169)

[Txt2DbfConverter ( ForgeAgProgDataFormat & dataFormat, const char \* targetDir, WrongFormatAction actionTakenWhenWrong ) 48](#_Toc458693170)

[~Txt2DbfConverter () 49](#_Toc458693171)

[ConversionResult Txt2DbfConverter::Convert ( ProgressCallBack \* cb ) 49](#_Toc458693172)

[void Txt2DbfConverter::ConvertAsync ( ProgressCallBack \* cb ) 50](#_Toc458693173)

[void Txt2DbfConverter::CancelAsync () 50](#_Toc458693174)

[Txt2PackedDbfConverter Class Reference 53](#_Toc458693175)

[Txt2PackedDbfConverter ( BasicProgDataFormat & dataFormat, const char \* target, WrongFormatAction actionTakenWhenWrong ) 53](#_Toc458693176)

[Txt2PackedDbfConverter ( CProgDataFormat & dataFormat, const char \* target, WrongFormatAction actionTakenWhenWrong ) 53](#_Toc458693177)

[Txt2PackedDbfConverter ( ForgeAgProgDataFormat &dataFormat, const char \* target, WrongFormatAction actionTakenWhenWrong ) 54](#_Toc458693178)

[Txt2PackedDbfConverter ( BasicProgDataFormat \* dataFormatList, int n, const char \* target, WrongFormatAction actionTakenWhenWrong ) 54](#_Toc458693179)

[Txt2PackedDbfConverter ( CProgDataFormat \* dataFormatList, int n, const char \* target, WrongFormatAction actionTakenWhenWrong ) 54](#_Toc458693180)

[Txt2PackedDbfConverter ( ForgeAgProgDataFormat \* dataFormatList, int n, const char \* target, WrongFormatAction actionTakenWhenWrong ) 55](#_Toc458693181)

[ConversionResult Txt2PackedDbfConverter::Convert ( ProgressCallBack \* cbPtr ) 55](#_Toc458693182)

[void Txt2PackedDbfConverter::ConvertAsync ( ProgressCallBack \* cb ) 57](#_Toc458693183)

[void Txt2PackedDbfConverter::CancelAsync () 57](#_Toc458693184)

# Introduction

Data Converter C++ API is a set of functions that performs file format conversion. Currently, the API supports text

file to CipherLab 8-Series Mobile Computer RAM format, text to CipherLab 8-Series Mobile Computer SD card

format, and CipherLab 8-Series Mobile Computer SD card to text format conversion.

This document is in compliance with Data Converter C++ API version 2.1.0 and above.

# How to create a C++ console project in Windows

1. Using Visual Studio IDE to create a C++ console project

**File** > **New** > **Project** > **Visual C++** > **Win32 Console Application**



**Figure 2.1: Create Console Application**

1. Unzip the DataConverter\_CPP\_API.zip

**There are three folders inside the zip file:**

**\include** Bunch of header files.

**\lib** CipherLab.DataConverter.lib: Data Converter API library

CipherLab.DataConverter.dll: Data Converter API dll

**\doc** DataConverter\_CPP\_API\_vX.X.pdf: User Guide

1. Copy both "\include" and "\lib" folder into your project directory
2. Add library header directory

**Project** > **Properties** >

**Configuration Properties** > **C/C++** > **General** > **Additional Include Directories**

Add the "\include" folder



Figure 2.2: General Setting

1. Add link library

**Project** > **Properties** >

**Configuration Properties** > **Linker** > **General** > **Additional Libary Directories**

Add the "\lib" folder



Figure 2.3: Linker Directory

**Project** > **Properties** > **Configuration Properties** > **Linker** > **input** > **Additional Dependencies**

Add "CipherLab.DataConverter.lib"



Figure 2.4: Linker Library

1. Put CipherLab.DataConverter.dll in the same directory of program's execution file
2. In the main program, you could simply include "DataConverter.h" which contains all available headers.

# How to create a C++ Console Project in Linux

## For amd64:

1. Using Code::Block IDE to create a C++ console project

File -> New -> Project -> Console application



1. Copy both "\include" and "\lib" folder into your project directory
2. Add Compiler other options

Project -> Build options -> Compliler settings -> Other options

* -lpthread
* -lm



1. Add Compiler defined variable

Note: Support GNU C++ compiler version 5

Project -> Build options -> Compliler settings -> #defines

* \_GLIBCXX\_USE\_CXX11\_ABI=0



1. Add link library to build options

Project -> Build options -> Linker settings

* Add the “Linux amd64/libDataConverter.a” which is inside the lib folder.
* Add “libpthread.so” which is inside the “/usr/lib/x86\_64-linux-gnu” folder



1. Add Compiler search directories

Project -> Build options -> Search directories -> Compiler

Add the “include” folder



1. In the main program, you could simply include “DataConverter.h” which contains all available headers.

## For i386

1. Using Code::Block IDE to create a C++ console project

File -> New -> Project -> Console application





1. Copy both "\include" and "\lib" folder into your project directory
2. Add Compiler other options

Project -> Build options -> Compliler settings -> Other options

* -lpthread
* -lm



1. Add Compiler defined variable

Note: Support GNU C++ compiler version 5

Project -> Build options -> Compliler settings -> #defines

* \_GLIBCXX\_USE\_CXX11\_ABI=0



1. Add link library to build options

Project -> Build options -> Linker settings

* Add the “Linux i386/libDataConverter.a” which is inside the lib folder.
* Add “libpthread.so” which is inside the “/usr/lib/i386-linux-gnu” folder



1. Add Compiler search directories

Project -> Build options -> Search directories -> Compiler

Add the “include” folder



1. In the main program, you could simply include “DataConverter.h” which contains all available headers.

Sample Code

|  |
| --- |
| #include "DataConverter.h"  #include <iostream>  #include <stdio.h>  #include <cstdlib>  #if defined \_WIN32 || defined \_WIN64  #else  #include <unistd.h>  #endif  using namespace std;  void progressCallBack\_dbf2txt(float percent, int recordCount, const char\* status)  {  printf("Percentage: %.02f ", percent);  printf("RecordCount: %d ", recordCount);  printf("Status: %snn", status);  }  int main()  {  ProgressCallBack\* cbPtr = new ProgressCallBack();  cbPtr->SetCallBack(progressCallBack\_dbf2txt);  #if defined \_WIN32 || defined \_WIN64  string srcC = "D://DC//testing folder//txt2dbf\_c//input4.DB0";  string tarC = "D://DC//testing folder//dbf2txt\_c//input4.TXT";  #else  string srcC = "/home/DC/testing folder/txt2dbf\_c/input\_4f.DB0";  string tarC = "/home/DC/testing folder/dbf2txt\_c/nput4.TXT";  #endif  Dbf2TxtConverter\* d2tObj = new Dbf2TxtConverter(C, srcC.c\_str(), tarC.c\_str());  try{  d2tObj->ConvertAsync(cbPtr);  d2tObj->CancelAsync();  ConversionResult\* result = d2tObj->result;  cout << "status=" << endl;  switch(result->status)  {  case Failed:  cout << "Failed" << endl;  break;  case Cancelled:  cout << "Cancelled" << endl;  break;  #if defined \_WIN32 || defined \_WIN64  case **CipherLab::DataConverter::Unknown:**  #else  case Unknown:  #endif  cout << "Unknown" << endl;  break;  case SucceededWithRecordsSkipped:  cout << "SucceededWithRecordsSkipped" << endl;  break;  case Succeeded:  cout << "Succeeded" << endl;  break;  default:  cout << "Totally Unknown" << endl;  }  cout << "Output File Paths: " << endl;  for(int i = 0; i < result->getNumberOfOutputFiles(); i++)  {  cout << i << ") " << result->getOutputFilePaths(i) << endl;  }  cout << "ErrorRecord: " << endl;  for(int i = 0; i < result->getNumberOfErrorRecords(); i++)  {  ErrorRecord e = result->getErrorRecord(i);  cout << i << "): ";  cout << "filename = " << e.getFileName() << endl;  cout << e.getRecordNum() << endl;  cout << e.getRecordText() << endl;  }  }catch (formatException& e)  {  cerr << e.getMessage() << endl;  getchar();  exit(0);  }  #if defined \_WIN32 || defined \_WIN64  cin.get();//pause console to see the message  #else  sleep(10);  #endif  return 0;  } |

# Namespace

## CipherLab::DataConverter

the namespace of Data Converter C++ API

# Enumerations

### enum AGXType

The AGXType enumeration defines the type of **AGX** format for 8-series.

**Enumerator**

**Series86** The agx format type for 8600 model.

**Series86\_Encrypted** The agx format type for 8600 model. The agx format type for 8600 model which is generated by Forge AG v2.0 and later.

**Others** The agx format type for 8-series except of 8600 model.

**Others\_Encrypted** The agx format type for 8-series except of 8600 model. The agx format type for 8600 model which is generated by Forge AG v2.0 and later**.**

### enum BasicProgDbfFileName

The BasicProgDbfFileName enumeration defines the DBF file names for Basic programming record format.

**Enumerator**

**F1** The filename part of the DBF file is F1.

**F2** The filename part of the DBF file is F2.

**F3** The filename part of the DBF file is F3.

**F4** The filename part of the DBF file is F4.

**F5** The filename part of the DBF file is F5.

### enum ConversionStatus

The ConversionStatus enumeration defines the current status of conversion process.

**Enumerator**

**Failed** Conversion status is failed.

**Cancelled** Conversion status is cancelled.

**Unknown** Conversion status is unknown.

**SucceededWithRecordsSkipped** Conversion status is succeeded which might have some records skipped.

**Succeeded** Conversion status is succeeded.

### enum FileType

The FileType enumeration defines the type of file for doing Dbf to PC conversion.

**Enumerator**

**UnKnown** Unknown file.

**Lookup** A lookup file.

**Transaction** A transaction file.

### enum ForgeAgProgDbfFileName

The ForgeAgProgDbfFileName enumeration defines the type of lookup file for Forge AG programming record format.

**Enumerator**

**FirstLookup** First lookup file.

**SecondLookup** Second lookup file.

**ThirdLookup** Third lookup file.

### enum ProgrammingTool

The ProgrammingTool enumeration defines the programming tool which the user uses to program his/her application.

**Enumerator**

**C** C programming tool.

**Basic** Basic programming tool.

**ForgeAg** Forge Ag programming tool.

### enum WrongFormatAction

The WrongFormatAction enumeration defines the actions to take when encountering wrong data format during

conversion.

**Enumerator**

**Stop** Stop the conversion process when data format is not correct.

**Reformat** Reformat the data record when the format is not correct.

**Skip** Skip the current record and continue with the next record.

# Class Documentation

## AGX Class Reference

**Member Functions**

### static int DetermineFormatType (const char \*path)

To determine the Agx file type.

**Parameters**

|  |  |
| --- | --- |
| path | **AGX** file path |

**Return Value**

|  |  |
| --- | --- |
| int | [AGXType](#_enum_AGXType) or -1 to indicate not able to open file |

**Example**

|  |
| --- |
| string path2 = "D:\\DC\\testing folder\\agx\\8600\_Basic\_Test1.AGX";  int val2 = AGX::DetermineFormatType(path2.c\_str());  cout << "fileType = " ;  switch(val2)  {  case CipherLab::DataConverter::Series86:  cout << "Series86" << endl;  break;  case CipherLab::DataConverter::Series86\_Encrypted:  cout << "Series86\_Encrypted" << endl;  break;  case CipherLab::DataConverter::Others:  cout << "Others" << endl;  break;  case CipherLab::DataConverter::Others\_Encrypted:  cout << "Others\_Encrypted" << endl;  break;  default:  cout << "Cannot open file" << endl;  } |

### static ForgeAgProgDataFormat \* Read (const char \* path, int &counter)

Reads record format details from **AGX** file.

**Parameters**

|  |  |
| --- | --- |
| path | **AGX** file path |
| counter | The number of records that references the number records of [**ForgeAgProgDataFormat**](#_ForgeAgProgDataFormat_Class_Referen) |

**Return Value**

|  |  |
| --- | --- |
| [**ForgeAgProgDataFormat**](#_ForgeAgProgDataFormat_Class_Referen) **\*** | Point to an array of [**ForgeAgProgDataFormat**](#_ForgeAgProgDataFormat_Class_Referen) |

**Example**

|  |
| --- |
| string path = "D:\\DC\\testing folder\\agx\\82\_fixed.AGX";  int numOfLookup = 0;  ForgeAgProgDataFormat \*for86df = AGX::Read(path2.c\_str(), numOfLookup);  for(int i = 0; i < numOfLookup; i++)  {  cout << "[" << i << "] DbfFileName: " << for86df[i].DbfFileName << endl;  cout << for86df[i].actionTakenWhenFormatIsWrong << endl;  for(int j= 0; j < for86df[i].getNumberOfRecordFields(); j++)  {  RecordField rec = for86df[i].GetRecordFieldAt(j);  cout << "[" << j << "] Length(" << rec.Length << ") Offset(" << rec.Position << ") Key(" << rec. IsKey << ")" << endl;  }  } |

## ApiVersion Class Reference

**Detailed Description**

Gets current API version info (major version . minor version . build version)

**Member Functions**

### int GetApi\_major\_version ( )

Get API major number.

**Return Value**

|  |  |
| --- | --- |
| int | API major number |

### int GetApi\_minor\_version ( )

Get API minor number.

**Return Value**

|  |  |
| --- | --- |
| int | API minor number |

### int GetApi\_build\_version ( )

Get API build number.

**Return Value**

|  |  |
| --- | --- |
| int | API build number |

### const char \* GetApiVersion ( )

Get API version info

**Return Value**

|  |  |
| --- | --- |
| const char\* | API version |

## BasicProgDataFormat Class Reference

Inheritance diagram

**Detailed Description**

Defines record format for Basic programming conversions.

**Remarks:**

The maximum number of record fields for both non-8600 model and 8600 model is 8.

Txt2PackedDbf conversion: For non-8600 model, the format can have 1~3 key fields specified; however, for 8600

model, the format can have 0~5 key fields specified.

Txt2Dbf conversion: For non-8600 model, the format can have 0~3 key fields specified; however, for 8600 model,

the format can have 0~5 key fields specified.

**Member Functions**

### BasicProgDataFormat ()

Default Constructor of **BasicProgDataFormat** class.

### ~BasicProgDataFormat ()

Destructor of **BasicProgDataFormat** Class.

### BasicProgDataFormat ( const char \* path, [BasicProgDbfFileName](#_Enum_BasicProgDbfFileName) dbfFileName )

Constructor of BasicDataFormat Class.

**Parameters**

|  |  |
| --- | --- |
| *path* | *the input file path* |
| *dbfFileName* | *the output DBF file name; enumerator of* [*BasicProgDbfFileName*](#_Enum_BasicProgDbfFileName) |

### BasicProgDataFormat ( const char \* path, [BasicProgDbfFileName](#_Enum_BasicProgDbfFileName) dbfFileName, char delimiter )

Constructor of BasicDataFormat Class.

**Parameters**

|  |  |
| --- | --- |
| *path* | *the input file path* |
| *dbfFileName* | *the output DBF file name; enumerator of* [*BasicProgDbfFileName*](#_Enum_BasicProgDbfFileName) |
| *delimiter* | *the delimiter character that is used in the record to separate data.* |

**Public Attributes**

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| [BasicProgDbfFileName](#_Enum_BasicProgDbfFileName) | DbfFileName | The output DBF file name of BasicProgDataFormat. |

## ConversionResult Class Reference

**Detailed Description**

Gets the post conversion result including the possible **ErrorRecord** list and the output file paths.

**Member Functions**

• ErrorRecord getErrorRecord (int index)

Get the error record with specific index.

**Parameters**

|  |  |
| --- | --- |
| index | the index of ErrorRecord vector |

**Return Value**

|  |  |
| --- | --- |
| **ErrorRecord** | the error record with specific index |

### int getNumberOfErrorRecords ()

Get the number of Error Records.

**Return Value**

|  |  |
| --- | --- |
| *int* | *the number of Error Records* |

### const char \* getOutputFilePaths (int index)

Get the output file's path with specific index.

**Parameters**

|  |  |
| --- | --- |
| *index* | *the index of ErrorRecord vector* |

**Return Value**

|  |  |
| --- | --- |
| *const char\** | *output file’s path with specific index.* |

### int getNumberOfOutputFiles ()

Get the number of the output files.

**Return Value**

|  |  |
| --- | --- |
| *int* | *the number of the output files.* |

### ConversionResult ()

Constructor of **ConversionResult** class.

### ~ ConversionResult ()

Destructor of **ConversionResult** class.

**Public Attributes**

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| [***ConversionStatus***](#_enum_ConversionStatus) | status | *the defined type of* [*ConversionStatus*](#_enum_ConversionStatus) *enumeration.* |
| *static std::vector<* [***ErrorRecord***](#_ErrorRecord_Class_Reference) *>* | errorRecordList | *the vector of* [*ErrorRecord*](#_ErrorRecord_Class_Reference) *objects.* |
| *static std::vector< string >* | outputFilePaths | *the vector of output files’ path.* |

## CProgDataFormat Class Reference

Inheritance diagram

**Detailed Description**

**CProgDataFormat** class.

Defines record format for C programming conversions.

**Remarks:**

The maximum number of record fields for both non-8600 model and 8600 model is 8.

Txt2PackedDbf conversion: For non-8600 model, the format can have 1~8 key fields specified; however, for 8600

model, the format can have 0~8 key fields specified.

Txt2Dbf conversion: For both non-8600 model and 8600 model, the format can have 0~8 key fields specified.

**Member Functions**

### CProgDataFormat ()

Constructor of **CProgDataFormat** class.

### ~CProgDataFormat ()

Destructor of **CProgDataFormat** class.

### CProgDataFormat ( const char \*path, const char \* dbfFileName )

Constructor of **CProgDataFormat** Class.

**Parameters**

|  |  |
| --- | --- |
| *path* | *the input file path* |
| *dbfFileName* | *the output DBF file name* |

### CProgDataFormat ( const char \* path, const char \* dbfFileName, char delimiter )

Constructor of **CProgDataFormat** Class.

**Parameters**

|  |  |
| --- | --- |
| *path* | *the input file path* |
| *dbfFileName* | *the output DBF file name* |
| *delimiter* | *the delimiter character that is used in the record to separate data.* |

**Public Attributes**

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| const char\* | DbfFileName | The output DBF file name of CProgDataFormat. |

## DataFormat Class Reference

Inheritance diagram

**Member Functions**

### DataFormat (int maxFields, char delimiter=0x00)

Constructor of **DataFormat** class.

**Parameters**

|  |  |
| --- | --- |
| *maxFields* | *the maximum number of fields in the data format.* |
| *delimiter* | *the delimiter character that is used in the record to separate data.* |

### ~DataFormat ()

Destructor of **DataFormat** class.

### const char \*getPath ()

Get the input file path.

**Return Value**

|  |  |
| --- | --- |
| *const char\** | *input file’s path.* |

### void setPath (const char \*path)

Set the input file path.

**Parameters**

|  |  |
| --- | --- |
| *path* | *the input file path* |

### char getDelimiter ()

Get the delimiter character.

**Return Value**

|  |  |
| --- | --- |
| *char* | *the delimiter character.* |

### void setDelimiter (char ch)

Set the delimiter character.

**Parameters**

|  |  |
| --- | --- |
| *ch* | *the delimiter character* |

### bool isRecordDelimited ()

Indicate **whether the record uses a delimiter character to separate data.**

**Return Value**

|  |  |
| --- | --- |
| *bool* | ***whether the record uses a delimiter character to separate data.*** |

### const char \* getDbfFileName ()

Get the output DBF file name.

**Return Value**

|  |  |
| --- | --- |
| *const char\** | *the output DBF file name.* |

### void setDbfFileName (const char \* fileName)

Set the output DBF file name.

**Parameters**

|  |  |
| --- | --- |
| *filename* | *set the output DBF file name.* |

### int getNumberOfRecordFields ()

Get the number of recordfields in the data format.

**Return Value**

|  |  |
| --- | --- |
| *bool* | *the number of record fields in the data format* |

### int getMaxRecordFields ()

Get the maximum record fields in the data format.

**Return Value**

|  |  |
| --- | --- |
| *int* | *the maximum record fields in the data format.* |

### int getKeyCount ()

Get the number of key fields in the data format.

**Return Value**

|  |  |
| --- | --- |
| *bool* | ***whether the record uses a delimiter character to separate data.*** |

### RecordField GetRecordFieldAt (int index)

Gets a record field from the specified location.

**Parameters**

|  |  |
| --- | --- |
| *index* | *specified location* |

**Return Value**

|  |  |
| --- | --- |
| *RecordField* | ***the record field from the specified location.*** |

### void EditRecordFieldPosition (int index, int position)

Edit specific record field's start position.

**Parameters**

|  |  |
| --- | --- |
| *index* | *the index of the record field* |
| *position* | *edit the start position of the record field* |

### void EditRecordFieldLength (int index, int length)

Edit specific record field's length.

**Parameters**

|  |  |
| --- | --- |
| *index* | *the index of the record field* |
| *position* | *edit the length of the record field* |

### void EditRecordFieldIsKey (int index, bool isKey)

Edit specific record field whether be a key field.

**Parameters**

|  |  |
| --- | --- |
| *index* | *the index of the record field* |
| *position* | *edit the specific record field whether be a key field or not* |

### void ClearRecordFields ()

Clear the record fields in a data format.

### void AddRecordField (RecordField recordField)

Adds a record field to the **DataFormat** class.

**Parameters**

|  |  |
| --- | --- |
| *recordField* | *an* [*RecordField*](#_RecordField_Class_Reference) *object* |

### void ParseRecordFields (const char \* inputFilePath)

Parse the input file to generate the vector of record fields.

**Parameters**

|  |  |
| --- | --- |
| *inputFilePath* | *the input file path* |

## Dbf2TxtConverter Class Reference

**Dbf2TxtConverter** class.

Converts DBF file or transaction (DAT) file on SD card to text file.

**Public Member Functions**

### Dbf2TxtConverter (ProgrammingTool tool, const char \*source, const char \* target)

**Initializes a new instance of the Dbf2TxtConverter class with specific programming language, input file path, and the output file path.**

**Parameters**

|  |  |
| --- | --- |
| *tool* | *the* [*ProgrammingTool*](#_enum_ProgrammingTool) *enumeration defines the programming tool which the user uses to program his/her application.* |
| *source* | *the path of DBF file or transaction (DAT) file* |
| *target* | *the path of output* |

### void Convert (ProgressCallBack \*cb)

Requests action of conversion.

**Parameters**

|  |  |
| --- | --- |
| *cb* | *a function point to user defined progress report function. If there is no need to report progress, set cb to NULL.* |

### void ConvertAsync ( ProgressCallBack \* cb )

Requests action of asynchronous conversion.

**Parameters**

|  |  |
| --- | --- |
| *cb* | *a function point to user defined progress report function. If there is no need to report progress, set cb to NULL.* |

### void CancelAsync ()

Requests cancellation of asynchronous conversion.

**Public Attributes**

|  |  |
| --- | --- |
| **ConversionResult** \* **result** | point to the **ConversionResult** object for conversion result |

**Example**

|  |
| --- |
| #include "DataConverter.h"  #include <iostream> //for cin, cout and endl  #include <string.h>  #include <stdio.h>  #include <cstdlib>  using namespace std;  using namespace CipherLab::DataConverter;  void progressCallBack(float percent, int recordCount, const char\* status)  {  printf("Percentage: %.02f ", percent);  printf("RecordCount: %d ", recordCount);  printf("Status: %s\\", status);  }  int main()  {  ProgressCallBack\* cbPtr = new ProgressCallBack();  cbPtr->SetCallBack(progressCallBack);  string srcC = "D:\\DC\\testing folder\\txt2dbf\_c\\input4.DB0";  string tarC = "D:\\DC\\testing folder\\dbf2txt\_c\\input4.TXT";  Dbf2TxtConverter\* d2tObj = new Dbf2TxtConverter(C, srcC.c\_str(), tarC.c\_str());  try{  d2tObj->ConvertAsync(cbPtr);  d2tObj->CancelAsync();  }catch (formatException& e)  {  cerr << e.getMessage() << endl;  getchar();  exit(0);  }  cin.get();//pause console to see the message  return 0;  } |

## ErrorRecord Class Reference

**Detailed Description**

object which contains the error info when detecting wrong data format during conversion

**Public Member Functions**

### ErrorRecord ( string fileName, long recordNum, string recordText, bool isSkip )

**ErrorRecord** Constructor.

**Parameters**

|  |  |
| --- | --- |
| *fileName* | *the input file name* |
| *recordNum* | *identifies the sequence of the record* |
| *recordText* | *identifies the text of record.* |
| *isSkip* | *identifies whether the action to take when detecting wrong data format during conversion is 'skip' or not.* |

### ErrorRecord ()

Default Constructor of **ErrorRecord**

### ~ErrorRecord ()

Destructor of **ErrorRecord**

### const char \* getFileName ()

Get the input file name.

**Return Value**

|  |  |
| --- | --- |
| *const char\** | *the input file name* |

### **long getRecordNum ()**

Get the sequence of the record.

**Return Value**

|  |  |
| --- | --- |
| *long* | *sequence of the record* |

### const char \* getRecordText ()

Get the text of record.

**Return Value**

|  |  |
| --- | --- |
| *const char\** | *the text of record* |

### bool isSkipped ()

determine whether the **ErrorRecord** is skipped

**Return Value**

|  |  |
| --- | --- |
| *bool* | *determine whether the* ***ErrorRecord*** *is skipped* |

## ForgeAgProgDataFormat Class Reference

Inheritance diagram

**Detailed Description**

Defines record format for **ForgeAgProgDataFormat** programming conversions.

**Remarks**:

The maximum number of record fields for non-8600 model is 8; however, the maximum number of record fields for 8600 model is 12.

For both Txt2PackedDbf conversion and Txt2Dbf conversion, the format must have one and only one key field

specified.

**Member Functions**

### ForgeAgProgDataFormat ()

Constructor of **ForgeAgProgDataFormat** class.

### ~ForgeAgProgDataFormat ()

Destructor of **ForgeAgDataFormat** class.

### ForgeAgProgDataFormat ( const char \*path, ForgeAgProgDbfFileName dbfFileName )

Constructor of **ForgeAgProgDataFormat** Class.

**Parameters**

|  |  |
| --- | --- |
| *path* | *the input file path* |
| *dbfFileName* | *the output DBF file name; enumerator of* [*ForgeAgProgDbfFileName*](#_enum_ForgeAgProgDbfFileName) |

### ForgeAgProgDataFormat ( const char \* path, ForgeAgProgDbfFileName dbfFileName, char delimiter )

Constructor of **ForgeAgProgDataFormat** Class.

**Parameters**

|  |  |
| --- | --- |
| *path* | *the input file path* |
| *dbfFileName* | *the output DBF file name; enumerator of* [*ForgeAgProgDbfFileName*](#_enum_ForgeAgProgDbfFileName) |
| *delimiter* | *the delimiter character that is used in the record to separate data.* |

**Public Attributes**

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| [ForgeAgProgDbfFileName](#_enum_ForgeAgProgDbfFileName) | DbfFileName | The output DBF file name of ForgeAgProgDataFormat. |

## formatException Class Reference

Inheritance diagram

**Detailed Description**

Exception class for DataConverter C++ API

**Public Member Functions**

### formatException (void)

Default constructor of **formatException** Class.

### formatException (string message, string info="")

Constructor of **formatException** Class.

**Parameters**

|  |  |
| --- | --- |
| *message* | *exception message* |
| *info* | *extra info of exception* |

### ~formatException (void) throw ()

Destructor of **formatException** Class.

### string getMessage ()

Return the runtime exception message.

**Return Value**

|  |  |
| --- | --- |
| *string* | *the runtime exception message with extra info if available.* |

## RecordField Class Reference

**Detailed Description**

Store record format information

**Public Member Data**

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| [int](#_enum_ConversionStatus) | Position | *the start position of the record field* |
| int | Length | the length of record field |
| bool | IsKey | whether the recordfield is a key field or not |

**Public Member Functions**

### **RecordField (int position, int length, bool isKey)**

**RecordField** constructor takes 3 parameters.

**Parameters**

|  |  |
| --- | --- |
| *position* | *the start position of the record field* |
| *length* | *the length of record field* |
| *isKey* | *whether the record field is a key field or not* |

### **RecordField (int position, int length)**

**RecordField** constructor takes 2 parameters.

**Parameters**

|  |  |
| --- | --- |
| *position* | *the start position of the record field* |
| *length* | *the length of record field* |

### RecordField ()

Default constructor of **RecordField** class.

## **ProgressCallBack Class Reference**

**Detailed Description**

for Convert() and ConvertAsync() functions use, enable to set a callback function to report the percentage, record count and status of conversion process.

### ProgressCallBack(void)

Default Constructor of **ProgessCallBack**

### void SetCallBack(CBFun fun)

Register *fn* as a callback function to be called automatically while conversion

**Parameters**

|  |  |
| --- | --- |
| *fun* | pointer to the function to be called. |

The CBFun member type is defined as:

*typedef void (\*CBFun)(float percentage, int recordCount, const char\* status);*

**Parameters**

|  |  |
| --- | --- |
| *percentage* | the current *percentage of process* |
| *recordCount* | the current processed record count |
| *status* | the processing status |

* Please check [Sample Code](#SampleCode) for reference.

## Txt2DbfConverter Class Reference

**Detailed Description**

**Converts lookup (.txt) file to DBF file for the device SD card.**

**Public Attributes**

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| [***ConversionStatus***](#_enum_ConversionStatus) | status | *the defined type of* [*ConversionStatus*](#_enum_ConversionStatus) *enumerator for conversion result.* |
| [*ConversionResult*](#_ConversionResult_Class_Reference) | result | *The* ***ConversionResult*** *object for conversion result.* |

**Public Member Functions**

### Txt2DbfConverter ( BasicProgDataFormat & dataFormat, const char \* targetDir, WrongFormatAction actionTakenWhenWrong )

Initializes a new instance of the **Txt2DbfConverter** class with basic programming data format.

**Parameters**

|  |  |
| --- | --- |
| *dataFormat* | *reference to a* ***BasicProgDataFormat*** *obj* |
| *targetDir* | *the directory path for the output file(s)* |
| *actionTakenWhenWrong* | *the action to take when detecting wrong data format during conversion* |

### Txt2DbfConverter ( CProgDataFormat & dataFormat, const char \* targetDir, WrongFormatAction actionTakenWhenWrong )

Initializes a new instance of the **Txt2DbfConverter** class with C programming data format.

**Parameters**

|  |  |
| --- | --- |
| *dataFormat* | *reference to a* ***CProgDataFormat*** *obj* |
| *targetDir* | *the directory path for the output file(s)* |
| *actionTakenWhenWrong* | *the action to take when detecting wrong data format during conversion* |

### Txt2DbfConverter ( ForgeAgProgDataFormat & dataFormat, const char \* targetDir, WrongFormatAction actionTakenWhenWrong )

Initializes a new instance of the **Txt2DbfConverter** class with forge ag programming data format.

**Parameters**

|  |  |
| --- | --- |
| *dataFormat* | *reference to a* ***ForgeAgProgDataFormat*** *obj* |
| *targetDir* | *the directory path for the output file(s)* |
| *actionTakenWhenWrong* | *the action to take when detecting wrong data format during conversion* |

### ~Txt2DbfConverter ()

Destructor of **Txt2DbfConverter** class.

### ConversionResult Txt2DbfConverter::Convert ( ProgressCallBack \* cb )

Requests conversion.

**Parameters**

|  |  |
| --- | --- |
| *cb* | *a function pointer to user defined progress report function. If there is no need to report progress, set cb to NULL.* |

**Return Value**

|  |  |
| --- | --- |
| [*ConversionResult*](#_ConversionResult_Class_Reference) | *point to the* ***ConversionResult*** *object for conversion result* |

**Example**

|  |
| --- |
| BasicProgDataFormat bpdf;  bpdf.setDelimiter(0x2C);  bpdf.DbfFileName = CipherLab::DataConverter::F2;  string path = "D:\\DcTestFolder\\stFiles\\D\\put\_delimiter\_15.txt";  bpdf.setPath(path.c\_str());  bpdf.ParseRecordFields(bpdf.getPath());  bpdf.EditRecordFieldIsKey(1, true);  bpdf.EditRecordFieldLength(2, 3);  bpdf.EditRecordFieldIsKey(2, true);  Txt2DbfConverter\* t2dObj = new Txt2DbfConverter(bpdf, "D:\\DcTestFolder\\PC2SD\\Delimiter\\API", Skip);  ConversionResult result;  try{  result = t2dObj->Convert(NULL);  }  catch(formatException& e)  {  cerr << e.getMessage() << endl;  getchar();  exit(0);  } |

### void Txt2DbfConverter::ConvertAsync ( ProgressCallBack \* cb )

Requests action of asynchronous conversion.

**Parameters**

|  |  |
| --- | --- |
| *cb* | *a function pointer to user defined progress report function. If there is no need to report progress, set cb to NULL.* |

### void Txt2DbfConverter::CancelAsync ()

Requests cancellation of asynchronous conversion.

**Example**

|  |
| --- |
| CProgDataFormat cpdf;  cpdf.setDelimiter(0x2C);  cpdf.AddRecordField(RecordField(1,4, false));  cpdf.AddRecordField(RecordField(6,2, true));  cpdf.AddRecordField(RecordField(9,3, false));  cpdf.DbfFileName = "input\_c";  string path = "D:\\DcTestFolder\\testFiles\\D\\input\_delimiter\_15.txt";  cpdf.setPath(path.c\_str());  for(int i = 0; i < cpdf.getNumberOfRecordFields(); i++)  {  RecordField a = cpdf.GetRecordFieldAt(i);  cout << "recordVector[" << i << "]" << " offset(" << a.Position << ") length(" << a.Length << ")" << endl;  }  Txt2DbfConverter\* t2dObj = new Txt2DbfConverter(cpdf, "D:\\DcTestFolder\\PC2SD\\Delimiter\\API", Skip);  try{  ConversionResult result;  ProgressCallBack\* cbPtr = new ProgressCallBack();  cbPtr->SetCallBack(progressCallBack);  t2dObj->ConvertAsync(cbPtr);  result = t2dObj->result;  switch(result.status)  {  case Failed:  cout << "Failed" << endl;  break;  case Cancelled:  cout << "Cancelled" << endl;  break;  case CipherLab::DataConverter::Unknown:  cout << "Unknown" << endl;  break;  case SucceededWithRecordsSkipped:  cout << "SucceededWithRecordsSkipped" << endl;  break;  case Succeeded:  cout << "Succeeded" << endl;  break;  default:  cout << "Totally Unknown" << endl;  }  cout << "Output File Paths: " << endl;  for(int i = 0; i < result.getNumberOfOutputFiles(); i++)  {  cout << i << ") " << result.getOutputFilePaths(i) << endl;  }  cout << "ErrorRecord: " << endl;  for(int i = 0; i < result.getNumberOfErrorRecords(); i++)  {  ErrorRecord e = result.getErrorRecord(i);  cout << i << "): ";  cout << "filename = " << e.getFileName() << endl;  cout << e.getRecordNum() << endl;  cout << e.getRecordText() << endl;  }  }  catch(formatException& e)  {  cerr << e.getMessage() << endl;  getchar();  exit(0);  } |

## **Txt2PackedDbfConverter Class Reference**

**Detailed Description**

**Converts lookup (.txt) file to DBF file for the device RAM.**

**Public Attributes**

|  |  |  |
| --- | --- | --- |
|  | Name | Description |
| [***ConversionStatus***](#_enum_ConversionStatus) | status | *the defined type of* [*ConversionStatus*](#_enum_ConversionStatus) *enumerator for conversion result.* |
| [*ConversionResult*](#_ConversionResult_Class_Reference) | result | *The* ***ConversionResult*** *object for conversion result.* |

**Public Member Functions**

### Txt2PackedDbfConverter ( BasicProgDataFormat & dataFormat, const char \* target, WrongFormatAction actionTakenWhenWrong )

Initializes a new instance of the **Txt2PackedDbfConverter** class with basic programming data format.

**Parameters**

|  |  |
| --- | --- |
| *dataFormat* | *reference to a* ***BasicProgDataFormat*** *obj* |
| *n* | *the number of dataformat* |
| *target* | *the output file path* |
| *actionTakenWhenWrong* | *the action to take when detecting wrong data format during conversion* |

### Txt2PackedDbfConverter ( CProgDataFormat & dataFormat, const char \* target, WrongFormatAction actionTakenWhenWrong )

Initializes a new instance of the **Txt2PackedDbfConverter** class with c programming data format.

**Parameters**

|  |  |
| --- | --- |
| *dataFormat* | *reference to a* ***CProgDataFormat*** *obj* |
| *n* | *the number of dataformat* |
| *target* | *the output file path* |
| *actionTakenWhenWrong* | *the action to take when detecting wrong data format during conversion* |

### Txt2PackedDbfConverter ( ForgeAgProgDataFormat &dataFormat, const char \* target, WrongFormatAction actionTakenWhenWrong )

Initializes a new instance of the **Txt2PackedDbfConverter** class with forgeAg programming data format.

**Parameters**

|  |  |
| --- | --- |
| *dataFormat* | *reference to a* ***ForgeAgProgDataFormat*** *obj* |
| *n* | *the number of dataformat* |
| *target* | *the output file path* |
| *actionTakenWhenWrong* | *the action to take when detecting wrong data format during conversion* |

### Txt2PackedDbfConverter ( BasicProgDataFormat \* dataFormatList, int n, const char \* target, WrongFormatAction actionTakenWhenWrong )

Initializes a new instance of the **Txt2PackedDbfConverter** (p. 41) class with multiple basic programming data format.

**Parameters**

|  |  |
| --- | --- |
| *dataFormatList* | *point to* ***BasicProgDataFormat*** *object(s) array* |
| *n* | *the number of dataformat* |
| *target* | *the output file path* |
| *actionTakenWhenWrong* | *the action to take when detecting wrong data format during conversion* |

### Txt2PackedDbfConverter ( CProgDataFormat \* dataFormatList, int n, const char \* target, WrongFormatAction actionTakenWhenWrong )

Initializes a new instance of the **Txt2PackedDbfConverter** class with multiple c programming data format.

**Parameters**

|  |  |
| --- | --- |
| *dataFormatList* | *point to* ***CProgDataFormat*** *object(s) array* |
| *n* | *the number of dataformat* |
| *target* | *the output file path* |
| *actionTakenWhenWrong* | *the action to take when detecting wrong data format during conversion* |

### Txt2PackedDbfConverter ( ForgeAgProgDataFormat \* dataFormatList, int n, const char \* target, [WrongFormatAction](#_enum_WrongFormatAction) actionTakenWhenWrong )

Initializes a new instance of the **Txt2PackedDbfConverter** class with multiple forgeAg programming data

formats.

**Parameters**

|  |  |
| --- | --- |
| dataFormatList | point to **ForgeAgProgDataFormat**  object(s) array |
| n | the number of dataformat |
| target | the output file path |
| actionTakenWhenWrong | the action to take when detecting wrong data format during conversion |

### ConversionResult Txt2PackedDbfConverter::Convert ( ProgressCallBack \* cbPtr )

Requests conversion.

**Parameters**

|  |  |
| --- | --- |
| *cbPtr* | *a function pointer to user defined progress report function. If there is no need to report progress, set cbPtr to NULL.* |

**Return Value**

|  |  |
| --- | --- |
| [*ConversionResult*](#_ConversionResult_Class_Reference) | *point to the* ***ConversionResult*** *object for conversion result* |

**Example**

|  |
| --- |
| BasicProgDataFormat bpdf = BasicProgDataFormat();  bpdf.setDelimiter(0x2C);  bpdf.DbfFileName = CipherLab::DataConverter::F1;  string path = "D:\\DcTestFolder\\testFiles\\D\\input\_delimiter\_15.txt";  bpdf.setPath(path.c\_str());  bpdf.ParseRecordFields(bpdf.getPath());  bpdf.EditRecordFieldIsKey(0, true);  bpdf.EditRecordFieldIsKey(1, true);  bpdf.EditRecordFieldIsKey(2, true);  for(int i = 0; i < bpdf.getNumberOfRecordFields(); i++)  {  RecordField a = bpdf.GetRecordFieldAt(i);  cout << "recordVector[" << i << "]" << " offset(" << a.Position << ") length(" << a.Length << ")" <  < endl;  }  Txt2PackedDbfConverter\* t2dObj = new Txt2PackedDbfConverter(bpdf, "D:\\DcTestFolder\\PC2SD\\Delimiter\\API\\output", Reformat);  try{  ConversionResult result;  ProgressCallBack\* cbPtr = new ProgressCallBack();  cbPtr->SetCallBack(progressCallBack\_txt2packedDbf);  result = t2dObj->Convert(cbPtr);  switch(result.status)  {  case Failed:  cout << "Failed" << endl;  break;  case Cancelled:  cout << "Cancelled" << endl;  break;  case CipherLab::DataConverter::Unknown:  break;  case SucceededWithRecordsSkipped:  cout << "SucceededWithRecordsSkipped" << endl;  break;  case Succeeded:  cout << "Succeeded" << endl;  break;  default:  cout << "Totally Unknown" << endl;  }  cout << "Output File Paths: " << endl;  for(int i = 0; i < result.getNumberOfOutputFiles(); i++)  {  cout << i << ") " << result.getOutputFilePaths(i) << endl;  }  for(int i = 0; i < result.getNumberOfErrorRecords(); i++)  {  if (i == 0)  cout << "ErrorRecord: " << endl;  ErrorRecord e = result.getErrorRecord(i);  cout << i << "): ";  cout << "filename = " << e.getFileName() << endl;  cout << e.getRecordNum() << endl;  cout << e.getRecordText() << endl;  }  }  catch(formatException& e)  {  cerr << e.getMessage() << endl;  getchar();  exit(0);  } |

### void Txt2PackedDbfConverter::ConvertAsync ( ProgressCallBack \* cb )

Requests action of asynchronous conversion.

**Parameters**

|  |  |
| --- | --- |
| *cb* | *a function ptr to user defined progress report function. If there is no need to report progress, set cb to NULL.* |

### void Txt2PackedDbfConverter::CancelAsync ()

Requests cancellation of asynchronous conversion.

**Example**

|  |
| --- |
| ForgeAgProgDataFormat fpdf = ForgeAgProgDataFormat();  fpdf.setDelimiter(0x2C);  fpdf.DbfFileName = CipherLab::DataConverter::FirstLookup;  string path = "D:\\DcTestFolder\\testFiles\\D\\input\_delimiter\_15.txt";  fpdf.setPath(path.c\_str());  fpdf.ParseRecordFields(fpdf.getPath());  fpdf.EditRecordFieldIsKey(0, true);  ForgeAgProgDataFormat fpdf1 = ForgeAgProgDataFormat();  fpdf1.setDelimiter(0x2C);  fpdf1.DbfFileName = CipherLab::DataConverter::FirstLookup;  path = "D:\\DcTestFolder\\testFiles\\D\\input\_delimiter\_15.txt";  fpdf1.setPath(path.c\_str());  fpdf1.ParseRecordFields(fpdf1.getPath());  fpdf1.EditRecordFieldIsKey(0, true);  ForgeAgProgDataFormat a[2];  a[0] = fpdf;  a[1] = fpdf1;  Txt2PackedDbfConverter\* t2dObj = new Txt2PackedDbfConverter(a, 2, "D:\\DcTestFolder\\PC2SD\\Delimiter\\API\\ForgeAg\\output", Reformat);  try{  ConversionResult result;  ProgressCallBack\* cbPtr = new ProgressCallBack();  cbPtr->SetCallBack(progressCallBack\_txt2packedDbf);  t2dObj->ConvertAsync(cbPtr);  result = t2dObj->result;  switch(result.status)  {  case Failed:  cout << "Failed" << endl;  break;  case Cancelled:  cout << "Cancelled" << endl;  break;  case CipherLab::DataConverter::Unknown:  cout << "Unknown" << endl;  break;  case SucceededWithRecordsSkipped:  cout << "SucceededWithRecordsSkipped" << endl;  break;  case Succeeded:  cout << "Succeeded" << endl;  break;  default:  cout << "Totally Unknown" << endl;  }  cout << "Output File Paths: " << endl;  for(int i = 0; i < result.getNumberOfOutputFiles(); i++)  {  cout << i << ") " << result.getOutputFilePaths(i) << endl;  }  for(int i = 0; i < result.getNumberOfErrorRecords(); i++)  {  if (i == 0)  cout << "ErrorRecord: " << endl;  ErrorRecord e = result.getErrorRecord(i);  cout << i << "): ";  cout << "filename = " << e.getFileName() << endl;  cout << e.getRecordNum() << endl;  cout << e.getRecordText() << endl;  }  }  catch(formatException& e)  {  cerr << e.getMessage() << endl;  getchar();  exit(0);  } |